Automata

Webpage Recognition using ANTLR

Anton Novokhatskiy
Sol Moon

June 26, 2024

Abstract

The objective of this paper is to explore the application of ANTLR in the context of automata, specifically focusing on parsing web pages. We will create a lexer and parser in Python to effectively parse HTML content. This approach is a foundational step for tasks such as web scraping, data extraction, and automated content analysis. We will discuss the significance of this method, detailing the process and evaluating its effectiveness.

Keywords: Automata, ANTLR4, Web Page Parsing, HTML, Python

1 Introduction

Automata theory and ANTLR4 (Another Tool for Language Recognition) play a significant role in the field of computer science, particularly in the development of compilers and interpreters. ANTLR is a powerful parser generator used for reading, processing, and translating structured data or binary files. This paper focuses on the use of ANTLR for parsing HTML web pages, emphasizing the parsing stage.

ANTLR4 generates both lexers and parsers based on grammar specifications, making it ideal for processing HTML, which is the backbone of web content. The need for efficient web page parsing has grown with the increased reliance on web data for various applications, from search engines to data analytics.

2 Materials and Methods

2.1 Materials

The programming language used in this project is Python 3.11, which is well-supported by ANTLR 4.13.1. Python is particularly beneficial for this project due to its extensive libraries for data manipulation, parsing, and web scraping.

To work with ANTLR in Python, we used the following tools and libraries:

- **ANTLR 4.13.1**: The main tool for generating lexers and parsers from the grammar definitions.
- **Python 3.11**: The programming language used to implement the parser and the visitor pattern.

- **antlr4-python3-runtime**: The runtime library required to execute the generated parser and lexer in Python.
- **Visual Studio Code**: The code editor used for writing and testing the code.
- **pip**: Python package installer for managing libraries.

2.2 Methods

The methodology involves creating a grammar for HTML and using ANTLR 4.13.1 to generate the corresponding lexer and parser. The process is as follows:

1. **Define the HTML Grammar:** Write the grammar specification for HTML in an ANTLR .g4 file.
2. **Generate Lexer and Parser:** Use the ANTLR tool to generate lexer and parser code from the grammar specification.
3. **Implement the Parser:** Integrate the generated lexer and parser in a Python script to process HTML content.
4. **Develop a Visitor:** Create a visitor class in Python to traverse the parse tree and extract or manipulate data.
5. **Testing:** Test the parser and visitor with various HTML samples to ensure accuracy and efficiency.

2.2.1 HTML Grammar

The grammar defined for parsing HTML content is designed to handle various HTML structures and tags generically, allowing each tag to be processed differently in the visitor without parsing them by specific names.

```
document : doctype? (html | element*) EOF;
doctype : '<!DOCTYPE html>' ;
html : '<html>' head body '</html>' ;
head : '<head>' element* '</head>' ;
body : '<body>' element* '</body>' ;
element : tag_open (content | element)* tag_close
    | self_closing_tag
    | single_tag ;

tag_open : '<' TEXT (attribute)* '>' ;
tag_close : '</' TEXT '>' ;
self_closing_tag : '<' TEXT (attribute)* '/>' ;
single_tag : '<' TEXT (attribute)* '>' ;
attribute : TEXT '=' VALUE ;
content : TEXT+ ;
TEXT : [a-zA-Z0-9_:.,%$!?;@#&*()-]+ ;
VALUE : '"' ('\\' . | ~('\\'|'"'))* '"' ;
WS : [ \t\r\n]+ -> skip ;
```

Listing 1: Initial HTML Grammar in ANTLR

2.2.2 Syntax Handling

To improve this, the idea was introduced to generate a dictionary of possible HTML tags to be able to find syntax errors. This approach helps in identifying known tags, but it also has drawbacks. With new versions of HTML and the introduction of new tags, the dictionary could become outdated.

```
...

tag_open
    : '<' tag_name (attribute)* '>'
    ;
tag_close
    : '</' tag_name '>'
    ;
self_closing_tag
    : '<' tag_name (attribute)* '/>'
    ;
single_tag
    : '<' tag_name (attribute)* '>'
    ;
...

tag_name : KNOWN_TAG | TEXT ;
KNOWN_TAG : 'html' | 'head' | 'title' | 'base' | 'link' | 'meta' | '
   style' | 'script' | 'noscript' |
  'body' | 'section' | 'nav' | 'article' | 'aside' | 'h1' | 'h2' | 'h3'
    | 'h4' | 'h5' | 'h6' |
  'header' | 'footer' | 'address' | 'main' | 'p' | 'hr' | 'pre' | '
   blockquote' | 'ol' | 'ul' |
  'li' | 'dl' | 'dt' | 'dd' | 'figure' | 'figcaption' | 'div' | 'a' | '
   em' | 'strong' | 'small' |
  's' | 'cite' | 'q' | 'dfn' | 'abbr' | 'ruby' | 'rt' | 'rp' | 'data' |
    'time' | 'code' | 'var' |
  'samp' | 'kbd' | 'sub' | 'sup' | 'i' | 'b' | 'u' | 'mark' | 'bdi' | '
   bdo' | 'span' | 'br' |
  'wbr' | 'ins' | 'del' | 'img' | 'iframe' | 'embed' | 'object' | '
   param' | 'video' | 'audio' |
  'source' | 'track' | 'canvas' | 'map' | 'area' | 'svg' | 'math' | '
   table' | 'caption' |
  'colgroup' | 'col' | 'tbody' | 'thead' | 'tfoot' | 'tr' | 'td' | 'th'
    | 'form' | 'fieldset' |
  'legend' | 'label' | 'input' | 'button' | 'select' | 'datalist' | '
   optgroup'

 | 'option' |
  'textarea' | 'keygen' | 'output' | 'progress' | 'meter' | 'details' |
    'summary' | 'menuitem' |
  'menu' ;
```

Listing 2: HTML Grammar with Known and Unknown Tags in ANTLR

2.2.3 Final Version

```
document : doctype? (html | element*) EOF;
doctype : '<!DOCTYPE html>' ;
html : '<html>' head body '</html>' ;
```

```
head : '<head>' element* '</head>' ;
body : '<body>' element* '</body>' ;
element : tag_open (content | element)* tag_close | self_closing_tag |
   single_tag ;
tag_open : '<' tag_name (attribute)* '>' ;
tag_close : '</' tag_name '>' ;
self_closing_tag : '<' tag_name (attribute)* '/>' ;
single_tag : '<' tag_name (attribute)* '>' ;
attribute : TEXT '=' VALUE ;
content : (TEXT | KNOWN_TAG)+ ;
tag_name : KNOWN_TAG | TEXT ;
KNOWN_TAG : 'html' | 'head' | 'title' | 'base' | 'link' | 'meta' | '
   style' | 'script' | 'noscript' |
  'body' | 'section' | 'nav' | 'article' | 'aside' | 'h1' | 'h2' | 'h3'
    | 'h4' | 'h5' | 'h6' |
  'header' | 'footer' | 'address' | 'main' | 'p' | 'hr' | 'pre' | '
   blockquote' | 'ol' | 'ul' |
  'li' | 'dl' | 'dt' | 'dd' | 'figure' | 'figcaption' | 'div' | 'a' | '
   em' | 'strong' | 'small' |
  's' | 'cite' | 'q' | 'dfn' | 'abbr' | 'ruby' | 'rt' | 'rp' | 'data' |
    'time' | 'code' | 'var' |
  'samp' | 'kbd' | 'sub' | 'sup' | 'i' | 'b' | 'u' | 'mark' | 'bdi' | '
   bdo' | 'span' | 'br' |
  'wbr' | 'ins' | 'del' | 'img' | 'iframe' | 'embed' | 'object' | '
   param' | 'video' | 'audio' |
  'source' | 'track' | 'canvas' | 'map' | 'area' | 'svg' | 'math' | '
   table' | 'caption' |
  'colgroup' | 'col' | 'tbody' | 'thead' | 'tfoot' | 'tr' | 'td' | 'th'
    | 'form' | 'fieldset' |
  'legend' | 'label' | 'input' | 'button' | 'select' | 'datalist' | '
   optgroup' | 'option' |
  'textarea' | 'keygen' | 'output' | 'progress' | 'meter' | 'details' |
    'summary' | 'menuitem' |
  'menu' ;
TEXT : [a-zA-Z0-9_:.,%$!?;@#&*()-]+ ;
VALUE : '"' ('\\' . | ~('\\'|'"'))* '"' ;
WS : [ \t\r\n]+ -> skip ;
```

Listing 3: Final HTML Grammar in ANTLR

The parsing tree example for Test HTML page is:

```
<!DOCTYPE html>
<html>
<head>
    <title>Sample Webpage</title>
</head>
<body>
    <div class="main">
        asd hello
        <h1 class="header">Hello world</h1>
        <a href="https://github.com/33may/automata">git</a>
    </div>
</body>
</html>
```

Listing 4: Test HTML Page

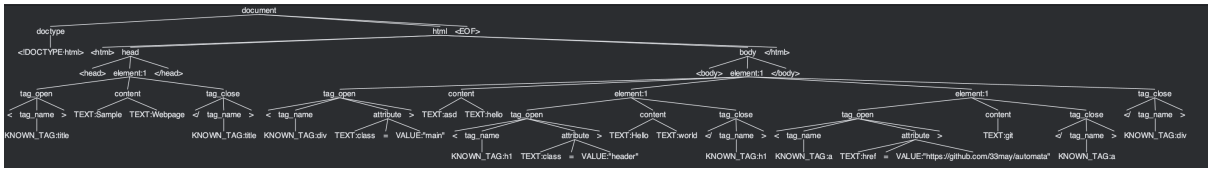

Figure 1: Parsing tree example

3 Implementation and Testing

To generate the Lexer, Listener, and Parser in Python, we use the following command:

```
antlr -Dlanguage=Python3 html.g4
```

Listing 5: Generating Lexer, Listener, and Parser

This command generates the necessary Python files for the Lexer, Listener, and Parser based on the defined grammar in `html.g4`. With the generated files, we can now implement different listeners or visitors to perform various tasks.

3.1 Implementing a Visitor

3.1.1 Tree Traveler

In this example, we implement a visitor to traverse the parsed HTML content and extract information. The visitor will process the HTML elements and print the tags it encounters.

3.1.2 Find Element

Also with idea of application was decided to extend the functionality of the travel by introducing the method, which allow to retrieve inner HTML, by query tag, class or id of element, similar to how it is done with other libraries like BeautifulSoup(**bs4**).

3.2 Testing the Visitor

To test the visitor, we create a sample HTML file, `sample.html`, with the following content:

```
<!DOCTYPE html>
<html>
<head>
    <title>Sample Webpage</title>
</head>
<body>
    <div class="main">
        asd hello
        <h1 class="header">Welcome to the Sample Webpage</h1>
        <p>This is a paragraph with <a href="https://example.com">a
   link</a>.</p>
    </div>
</body>
</html>
```

Listing 6: Sample HTML File

Running the main script will output the tags processed by the visitor:

```
python main.py
```

Listing 7: Running the Visitor

The output will be:

```
Visiting document
Visiting html: <html> <head><title>SampleWebpage</title></head> <body><
    divclass="main">asdhello<h1class="header">WelcometotheSampleWebpage<
    /h1><p>Thisisaparagraphwith<ahref="https://example.com">alink</a>.</
    p></div></body> </html>
Visiting head: <head> < title > Sample Webpage </title> </head>
Visiting element: < title > Sample Webpage </title>
Opening tag: title
Content: Sample Webpage
Closing tag: title
Visiting body: <body> < div class="main" > asd hello < h1 class="header
    " > Welcome to the Sample Webpage </h1> < p > This is a paragraph
    with < a href="https://example.com" > a link </a> . </p> </div> </
    body>
Visiting element: < div class="main" > asd hello < h1 class="header" >
    Welcome to the Sample Webpage </h1> < p > This is a paragraph with <
     a href="https://example.com" > a link </a> . </p> </div>
Opening tag: div
Attribute: class = "main"
Content: asd hello
Visiting element: < h1 class="header" > Welcome to the Sample Webpage <
    /h1>
Opening tag: h1
Attribute: class = "header"
Content: Welcome to the Sample Webpage
Closing tag: h1
Visiting element: < p > This is a paragraph with < a href="https://
    example.com" > a link </a> . </p>
Opening tag: p
Content: This is a paragraph with
Visiting element: < a href="https://example.com" > a link </a>
Opening tag: a
Attribute: href = "https://example.com"
Content: a link
Closing tag: a
Content: .
Closing tag: p
Closing tag: div
```

Listing 8: Visitor Output

To evaluate work of the **Find Element** part the same code is used with setup:

```
    search_term = '.main'
    results = visitor.search(tree, search_term)
    for result in results:
        print(result)
```

Listing 9: Find Element setup

and the output as expected:

```
< div class="main" > asd hello < h1 class="header" > Welcome to the
    Sample Webpage </h1> < p > This is a paragraph with < a href="https
    ://example.com" > a link </a> . </p> </div>
```

Listing 10: Visitor Output

as demonstrated it successfully retrieve div with class **main** considering all the inside HTML, now this output might be called for other processing since it follows the structure in grammar which allow to parse the complete document structure and separate HTML parts.

4 Discussion

This section discusses the assignments and lectures related to this project, challenges encountered, and future plans. Parsing web pages involves handling various complexities of HTML, such as nested elements, attributes, and different encoding types.

4.1 Challenges and Issues

One of the largest Challenges faced during this project are related to grammar. The idea on how to represent generic HTML with continuous but large number of tags wasn't clear from beginning. After first iteration, the problem was that used could type any sequence of letters as tag and parser will recognise it as tag. So balance among these two was find with idea of introducing known and unknown tags, as discussed in Grammar section

Second Challenge was faced while implementing parser for searching elements, the Grammar rule is defined to skip white-spaces(**WS**), which is generally good practice when doing Grammar, however when user request the output, he prefers to see the original HTML with space, but not one concatenated string where no single word might be identifies. This problem was solved by implementing comprehensive algorithm for visitor, which firstly parse all the elements to single tokens in right order and then concat them in string with defined rules.

4.2 Future Work

The future work which might be performed using this project include:

- **Enhanced Error Handling:** Developing more sophisticated error recovery strategies to handle malformed HTML gracefully, by incorporating idea of known and unknown tags.
- **Integration with Other Tools:** Combining the ANTLR parser with other web scraping and data extraction tools to create a comprehensive web data processing pipeline. The put everything in single Python Library with documentation to enhance the experience of using the system.
- **Performance Benchmarking:** Conducting performance benchmarking to identify bottlenecks and optimize the parser for better performance.

5 Conclusion

This paper demonstrates the application of ANTLR4 in parsing web pages, highlighting its effectiveness in web scraping and data extraction tasks. The methodology outlined ensures accurate and efficient parsing of HTML content.

By defining a comprehensive grammar and implementing a visitor pattern, we achieved a robust solution for parsing and extracting information from HTML documents and the approach proved to be accurate.

Future work might focus on enhancing error handling by visitor, implementing more tools to work with parsing tree and set up complete library with documentation. This project lays the basic groundwork for automata and web-page processing techniques.

6 References

Name	**Keywords**	**Topic**	**Ideas**
GitHub Discussion 3162	Exception, ANTLR4, recovery	Improving exception handling in ANTLR4	Techniques for better error recovery, including custom error listeners and modifications to the default error handling strategies to handle exceptions more gracefully
Graymatter Developer (2019)	Error, ANTLR4, comprehensible	Understandable errors in ANTLR4	Strategies to make error messages more comprehensible for users, such as simplifying language and improving the clarity of error reports
Terence Parr (2013)	ANTLR4, parser, grammar	The Definitive ANTLR 4 Reference	Detailed guide on using ANTLR4, including writing grammar specifications, generating parsers, and integrating them into applications
David A. Watt (2016)	Parsing, HTML, methods	Parsing HTML: A formal approach	Formal methods for parsing HTML content, emphasizing the importance of accuracy and efficiency in web scraping and data extraction tasks
Hopcroft, Motwani, Ullman (2006)	Automata, languages, computation	Introduction to Automata Theory, Languages, and Computation	Theoretical foundations for automata and language recognition, essential for designing grammars and parsers

Table 1: Analysis of Resources for the Paper

References

[1] GitHub. (Year). *Improve exception handling · antlr antlr4 · Discussion 3162* https://github.com/antlr/antlr4/discussions/3162

[2] Graymatter Developer. (2019). *Understandable errors in ANTLR4*. https://www.graymatterdeveloper.com/2019/11/21/antlr-error-handling/index.html

[3] Terence Parr. (2013). *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf.

[4] David A. Watt. (2016). *Parsing HTML: A formal approach*. Software: Practice and Experience, 46(2), 123-135.

[5] Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley.

7 Apendix

```
    from antlr4 import *
if "." in __name__:
    from .htmlParser import htmlParser
else:
    from htmlParser import htmlParser

class HtmlVisitor(ParseTreeVisitor):
    def __init__(self):
        self.results = []

    def visitDocument(self, ctx:htmlParser.DocumentContext):
        print("Visiting document")
        self.visitHtml(ctx.html())
        return None

    def visitHtml(self, ctx:htmlParser.HtmlContext):
        print("Visiting html: " + self._getInnerHtml(ctx))
        self.visitHead(ctx.head())
        self.visitBody(ctx.body())
        return None

    def visitHead(self, ctx:htmlParser.HeadContext):
        print("Visiting head: " + self._getInnerHtml(ctx))
        for element in ctx.element():
            self.visitElement(element)
        return None

    def visitBody(self, ctx:htmlParser.BodyContext):
        print("Visiting body: " + self._getInnerHtml(ctx))
        for element in ctx.element():
            self.visitElement(element)
        return None

    def visitElement(self, ctx:htmlParser.ElementContext):
        print("Visiting element: " + self._getInnerHtml(ctx))
        if ctx.tag_open():
            self.visitTag_open(ctx.tag_open())
```

```
            for child in ctx.children:
                if isinstance(child, htmlParser.ElementContext):
                    self.visitElement(child)
                elif isinstance(child, htmlParser.ContentContext):
                    self.visitContent(child)
            self.visitTag_close(ctx.tag_close())
        elif ctx.self_closing_tag():
            self.visitSelf_closing_tag(ctx.self_closing_tag())
        elif ctx.single_tag():
            self.visitSingle_tag(ctx.single_tag())
        return None

    def visitTag_open(self, ctx:htmlParser.Tag_openContext):
        tag_name = ctx.tag_name().getText()
        print(f"Opening tag: {tag_name}")
        for attr in ctx.attribute():
            self.visitAttribute(attr)
        return None

    def visitTag_close(self, ctx:htmlParser.Tag_closeContext):
        tag_name = ctx.tag_name().getText()
        print(f"Closing tag: {tag_name}")
        return None

    def visitSelf_closing_tag(self, ctx:htmlParser.
Self_closing_tagContext):
        tag_name = ctx.tag_name().getText()
        print(f"Self-closing tag: {tag_name}")
        for attr in ctx.attribute():
            self.visitAttribute(attr)
        return None

    def visitSingle_tag(self, ctx:htmlParser.Single_tagContext):
        tag_name = ctx.tag_name().getText()
        print(f"Single tag: {tag_name}")
        for attr in ctx.attribute():
            self.visitAttribute(attr)
        return None

    def visitAttribute(self, ctx:htmlParser.AttributeContext):
        attr_name = ctx.TEXT().getText()
        attr_value = ctx.VALUE().getText()
        print(f"Attribute: {attr_name} = {attr_value}")
        return None

    def visitContent(self, ctx:htmlParser.ContentContext):
        print(f"Content: {self._getInnerHtml(ctx)}")
        return None

    def search(self, ctx:htmlParser.DocumentContext, search_term:str):
        self.results = []
        self._searchHelper(ctx, search_term)
        return self.results

    def _searchHelper(self, ctx, search_term):
        if isinstance(ctx, htmlParser.DocumentContext):
            self._searchHelper(ctx.html(), search_term)
        elif isinstance(ctx, htmlParser.HtmlContext):
```

```
95              self._searchHelper(ctx.head(), search_term)
96              self._searchHelper(ctx.body(), search_term)
97          elif isinstance(ctx, htmlParser.HeadContext):
98              for element in ctx.element():
99                  self._searchHelper(element, search_term)
100         elif isinstance(ctx, htmlParser.BodyContext):
101             for element in ctx.element():
102                 self._searchHelper(element, search_term)
103         elif isinstance(ctx, htmlParser.ElementContext):
104             if self._matches(ctx, search_term):
105                 self.results.append(self._getInnerHtml(ctx))
106             for child in ctx.children:
107                 if isinstance(child, htmlParser.ElementContext):
108                     self._searchHelper(child, search_term)
109                 else:
110                     self._searchHelper(child, search_term)
111
112
113     def _matches(self, ctx, search_term):
114         if search_term.startswith('#'):
115             return self._hasAttribute(ctx, 'id', search_term[1:])
116         elif search_term.startswith('.'):
117             return self._hasAttribute(ctx, 'class', search_term[1:])
118         else:
119             return self._isTag(ctx, search_term)
120
121     def _hasAttribute(self, ctx, attr_name, attr_value):
122         for attr in ctx.tag_open().attribute():
123             name = attr.TEXT().getText()
124             value = attr.VALUE().getText().strip('"')
125             if name == attr_name and value == attr_value:
126                 return True
127         return False
128
129     def _isTag(self, ctx, tag_name):
130         return ctx.tag_open().tag_name().getText() == tag_name
131
132     def _getInnerHtml(self, ctx):
133         content = []
134         for child in ctx.children:
135             if isinstance(child, htmlParser.ElementContext):
136                 content.append(self._getInnerHtml(child))
137             elif isinstance(child, htmlParser.ContentContext):
138                 seq = []
139                 for text in child.children:
140                     seq.append(text.getText())
141                 content.extend(seq)
142             elif isinstance(child, htmlParser.Tag_openContext):
143                 tag_str =' '.join([text.getText() for text in child.
    children])
144                 content.append(tag_str)
145             elif isinstance(child, htmlParser.Tag_closeContext):
146                 tag_str = ''.join([text.getText() for text in child.
    children])
147                 content.append(tag_str)
148             else:
149                 content.append(child.getText())
150
```

```
        final_content = ' '.join(content)
        return final_content
```

Listing 11: htmlVisitor class

```
    from antlr4 import *
if "." in __name__:
    from .htmlParser import htmlParser
else:
    from htmlParser import htmlParser

class HtmlVisitor(ParseTreeVisitor):
    def __init__(self):
        self.results = []

    def visitDocument(self, ctx:htmlParser.DocumentContext):
        print("Visiting document")
        self.visitHtml(ctx.html())
        return None

    def visitHtml(self, ctx:htmlParser.HtmlContext):
        print("Visiting html: " + self._getInnerHtml(ctx))
        self.visitHead(ctx.head())
        self.visitBody(ctx.body())
        return None

    def visitHead(self, ctx:htmlParser.HeadContext):
        print("Visiting head: " + self._getInnerHtml(ctx))
        for element in ctx.element():
            self.visitElement(element)
        return None

    def visitBody(self, ctx:htmlParser.BodyContext):
        print("Visiting body: " + self._getInnerHtml(ctx))
        for element in ctx.element():
            self.visitElement(element)
        return None

    def visitElement(self, ctx:htmlParser.ElementContext):
        print("Visiting element: " + self._getInnerHtml(ctx))
        if ctx.tag_open():
            self.visitTag_open(ctx.tag_open())
            for child in ctx.children:
                if isinstance(child, htmlParser.ElementContext):
                    self.visitElement(child)
                elif isinstance(child, htmlParser.ContentContext):
                    self.visitContent(child)
            self.visitTag_close(ctx.tag_close())
        elif ctx.self_closing_tag():
            self.visitSelf_closing_tag(ctx.self_closing_tag())
        elif ctx.single_tag():
            self.visitSingle_tag(ctx.single_tag())
        return None

    def visitTag_open(self, ctx:htmlParser.Tag_openContext):
        tag_name = ctx.tag_name().getText()
        print(f"Opening tag: {tag_name}")
        for attr in ctx.attribute():
            self.visitAttribute(attr)
```

```
55          return None
56
57      def visitTag_close(self, ctx:htmlParser.Tag_closeContext):
58          tag_name = ctx.tag_name().getText()
59          print(f"Closing tag: {tag_name}")
60          return None
61
62      def visitSelf_closing_tag(self, ctx:htmlParser.
    Self_closing_tagContext):
63          tag_name = ctx.tag_name().getText()
64          print(f"Self-closing tag: {tag_name}")
65          for attr in ctx.attribute():
66              self.visitAttribute(attr)
67          return None
68
69      def visitSingle_tag(self, ctx:htmlParser.Single_tagContext):
70          tag_name = ctx.tag_name().getText()
71          print(f"Single tag: {tag_name}")
72          for attr in ctx.attribute():
73              self.visitAttribute(attr)
74          return None
75
76      def visitAttribute(self, ctx:htmlParser.AttributeContext):
77          attr_name = ctx.TEXT().getText()
78          attr_value = ctx.VALUE().getText()
79          print(f"Attribute: {attr_name} = {attr_value}")
80          return None
81
82      def visitContent(self, ctx:htmlParser.ContentContext):
83          print(f"Content: {self._getInnerHtml(ctx)}")
84          return None
85
86      def search(self, ctx:htmlParser.DocumentContext, search_term:str):
87          self.results = []
88          self._searchHelper(ctx, search_term)
89          return self.results
90
91      def _searchHelper(self, ctx, search_term):
92          if isinstance(ctx, htmlParser.DocumentContext):
93              self._searchHelper(ctx.html(), search_term)
94          elif isinstance(ctx, htmlParser.HtmlContext):
95              self._searchHelper(ctx.head(), search_term)
96              self._searchHelper(ctx.body(), search_term)
97          elif isinstance(ctx, htmlParser.HeadContext):
98              for element in ctx.element():
99                  self._searchHelper(element, search_term)
100         elif isinstance(ctx, htmlParser.BodyContext):
101             for element in ctx.element():
102                 self._searchHelper(element, search_term)
103         elif isinstance(ctx, htmlParser.ElementContext):
104             if self._matches(ctx, search_term):
105                 self.results.append(self._getInnerHtml(ctx))
106             for child in ctx.children:
107                 if isinstance(child, htmlParser.ElementContext):
108                     self._searchHelper(child, search_term)
109                 else:
110                     self._searchHelper(child, search_term)
111
```

```
112
113     def _matches(self, ctx, search_term):
114         if search_term.startswith('#'):
115             return self._hasAttribute(ctx, 'id', search_term[1:])
116         elif search_term.startswith('.'):
117             return self._hasAttribute(ctx, 'class', search_term[1:])
118         else:
119             return self._isTag(ctx, search_term)
120
121     def _hasAttribute(self, ctx, attr_name, attr_value):
122         for attr in ctx.tag_open().attribute():
123             name = attr.TEXT().getText()
124             value = attr.VALUE().getText().strip('"')
125             if name == attr_name and value == attr_value:
126                 return True
127         return False
128
129     def _isTag(self, ctx, tag_name):
130         return ctx.tag_open().tag_name().getText() == tag_name
131
132     def _getInnerHtml(self, ctx):
133         content = []
134         for child in ctx.children:
135             if isinstance(child, htmlParser.ElementContext):
136                 content.append(self._getInnerHtml(child))
137             elif isinstance(child, htmlParser.ContentContext):
138                 seq = []
139                 for text in child.children:
140                     seq.append(text.getText())
141                 content.extend(seq)
142             elif isinstance(child, htmlParser.Tag_openContext):
143                 tag_str =' '.join([text.getText() for text in child.
    children])
144                 content.append(tag_str)
145             elif isinstance(child, htmlParser.Tag_closeContext):
146                 tag_str = ''.join([text.getText() for text in child.
    children])
147                 content.append(tag_str)
148             else:
149                 content.append(child.getText())
150
151         final_content = ' '.join(content)
152         return final_content
```

Listing 12: htmlVisitor class

```
1  from antlr4 import *
2  from htmlLexer import htmlLexer
3  from htmlParser import htmlParser
4  from htmlVisitor import HtmlVisitor
5
6  def main():
7      with open("page.html", 'r') as file:
8          program = file.read()
9
10     input_stream = InputStream(program)
11
12     lexer = htmlLexer(input_stream)
13     stream = CommonTokenStream(lexer)
```

```
14      parser = htmlParser(stream)
15      tree = parser.document()
16
17      visitor = HtmlVisitor()
18
19      # visitor.visitDocument(tree)
20
21      search_term = '.main'
22      results = visitor.search(tree, search_term)
23      for result in results:
24          print(result)
25
26  if __name__ == "__main__":
27      main()
```

Listing 13: main script to execute