Archie

CS240H, Fall 2014

OmarDiab
osdiab@cs.stanford.edu

Alexander Atallah
aatallah@stanford.edu

Roshan Vidyashankar
rvid@ccrma.stanford.edu

June 12, 2014

Cascading Style Sheets are used to describe the look and feel of documents written in a markup language such as HTML. CSS consists of a set of rules that match parts of the document and apply styles to them. CSS has several limitations that make it painful for web developers. Archie tries to fix some of these limitations, particularly the non-dynamic nature of CSS.

1 Introduction

Cascading Style Sheets(CSS) are a language to describe the look and feel of a document. Their primary use is to separate document presentation from its content and structure. CSS styles are applied to documents written in a markup language such as HTML.

TODO: More introduction here

The rest of the paper is organized as follows. Section 2 provides some background on CSS and related technologies. In Section 3 we discuss our implementation. Section 4 talks about the future direction of our project followed by a conclusion.

2 Background

2.1 CSS Syntax

CSS has a very simple syntax. A style sheet consists a list of rules. Each rule-set consists of a selector and a declaration block.

```
h1{
   background: black;
   color: red;
}
```

Selectors are used to define which part of the markup a style applies to - elements of a spacific type, elements with a particular id or class or elements depending on their position in the DOM. A declaration block consists of a list of declarations in braces. Each declaration itself consists of a property and a value, separated by a colon. Multiple declarations are separated by a semicolon.

2.2 CSS Preprocessors

To deal with some of the limitations of CSS, many CSS preprocessors have gained popularity - SASS[1], Less[2] and Stylus[3]. They allow you to write stylesheets in a syntax similar but more expressive than CSS. They provide features like variables, nesting CSS rules, partials and mixins. These make stylesheets more modular and reusable thus allowing larger and more complex stylesheets.

2.3 Why Haskell?

There are several reasons why Haskell is an ideal language to implement this project in.

- We do not care about state
- Expresses context-free grammars (like the CSS Spec) naturally.
- Haskell's typechecking ensures inconsistent CSS is not allowed. For example, one should not be able to set a rule such as "width: auto px;"
- Haskell allows us to have composable functions that are more expressive than what current pre-processors offer.
- Monads can be used to express JS computation cleanly.

3 Design and Implementation

3.1 Generating CSS

We implemented the CSS 3 Specification from W3C.

3.2 Generating JavaScript

4 Future Work

While we have a basic working implementation, we had to cut several features in the interest of time. There are also a number of interesting directions the project could take. Some of the things to think about are discussed below.

4.1 Cleaner Syntax

In our basic implementation the code to generate CSS and JS is quite verbose. We have the choice to make the syntax look more like CSS or more like Haskell. Example syntax is provided below:

CSS-like Syntax:

```
#a {
    color: blue;
    width: {(width #b) / 2};
    height: {(height #c)};
}

#b {
    width: 50px;
}

#c {
    height: {(height #a)};
}
```

Haskell-like Syntax:

```
main = putCss $
  do p ? color red
     b ? color yellow
     article ?
       do strong ? background black
          abbr  <? fontVariant smallCaps
```

4.2 Incorporate Clay

Clay[4] is a CSS preprocessor like LESS and SASS, but implemented as an embedded domain specific language in Haskell. In Clay, all CSS selectors and style rules are first class Haskell functions which makes reuse and composability easy. Clay does a much better job at generating CSS than our current implementation since it has support for more complex selectors. We could choose to make Clay the default CSS generator in our project with the trade-off being more complexity.

Sample Clay Code:

```
import Clay

menu :: Css
menu = header |> nav ?
  do background    white
     color         "#04a"
     fontSize      (px 24)
     padding       20 0 20 0
     textTransform uppercase
     position      absolute
     left          0
     right         0
     bottom        (px (-72))
```

5 Conclusion

References

[1] SASS http://sass-lang.com/

[2] LESS http://lesscss.org/

[3] Stylus http://learnboost.github.io/stylus/

[4] Clay http://fvisser.nl/clay/