A Tool for Detecting and Refactoring the A?B*A Pattern in CSS.

An Executable Artefact for "The A?B*A Pattern: Undoing Style in CSS and Refactoring Opportunities it Presents"

@leonardpunt
Leonard Punt
University of Amsterdam, The Netherlands
Q42, The Netherlands

@sjoerdvisscher
Sjoerd Visscher
Q42, The Netherlands

@grammarware
Vadim Zaytsev
University of Amsterdam, The Netherlands
Raincode, Belgium

The tool can be download from the following web location:

- https://leonardpunt.github.io/masterproject/tool.zip

It is available under the MIT license.

Insightfulness

- **Timely**
  As noted in the paper, there is precious little research done on CSS. The complete list of all papers ever written on the subject and containing an explicit reference to it in the title, contains just 41 items [6]. ICSME 2016 will add two items to this list [1], [8].
- **Educating**
  Code smells are an important topic of research since at least the introduction of XP (before that they were usually called bad practices). It is not uncommon to see papers expanding the notion beyond code to architectures [3], spreadsheets [7], etc. Our main paper contributes to that trend by researching a stylesheet code smell and shows how previous research was insufficient [4]. The artefact enables replication of our results.

Usefulness

- **Purpose**
  CSS is a widely used language: the "preprocessors" are slowly gaining popularity, but still around half of development, evolution and maintenance is done on pure CSS — both according to developers survey [2] and results of our mining [5].
- **Awkwardness**
  Our tool that detects instantiations of the A?B*A pattern smell, especially when combined with similar tools detecting other smells, can be used in empirical research investigating presence of code smells in web applications. Without such tools, a project like that is doomed to concentrate on their development, but with them it can focus on comparatory analysis.
- **Cost**
  Our tool covers the pure detection part and does not implement any techniques to obtain all possible DOM states. That way it can be used as a module in a complete solution that would use crawling, machine learning, leverage test suite or do anything comparable.

Usability

- **Understandability**
  When built, the tool is available in both minified and non-minified form, the latter more suitable for inspection and comprehension. The way it runs, mimics the usual workflow of a web developer.
- **Tutorial**
  See the next page for a tutorial with checklists, recipes and examples.
- **Executability**
  A tool is distributed as a ZIP archive, and is easy to extract and build with `npm` and `grunt`. It runs in a browser with the help of Firebug plugin that is omnipresent on web developers' computers anyway.

References

[1] A. Charpentier, J.-R. Falleri, and L. Réveillère, "Automated Extraction of Mixins in Cascading Style Sheets," in *ICSME*, 2016, accepted, pending final version.
[2] C. Coyier, "CSS Tricks Poll Results: Popularity of CSS Preprocessors," https://css-tricks.com/poll-results-popularity-of-css-preprocessors/, 2012.
[3] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovi, "Toward a Catalogue of Architectural Bad Smells," in *QoSA*, ser. Lecture Notes in Computer Science, vol. 5581. Springer International Publishing, 2009, pp. 146–162.
[4] G. Gharachorlu, "Code Smells in Cascading Style Sheets: An Empirical Study and a Predictive Model." Master's thesis, University of British Columbia, Canada, 2014.
[5] B. Goncharenko and V. Zaytsev, "Expressing and Detecting Violations of Coding Conventions," 2016, submitted to SLE. Pending notification.
[6] ——, "Supplements for "Language Design and Implementation for the Domain of Coding Conventions"," https://dx.doi.org/10.6084/m9.figshare.3085831.v2, 2016.
[7] B. Jansen and F. Hermans, "Code Smells in Spreadsheet Formulas Revisited on an Industrial Dataset," in *ICSME*. IEEE, 2015, pp. 372–380.
[8] L. Punt, S. Visscher, and V. Zaytsev, "The A?B*A Pattern: Undoing Style in CSS and Refactoring Opportunities it Presents," in *ICSME*, 2016, accepted, pending final version.

Artefact Description

This tool detects undoing style in CSS code and is able to apply refactoring opportunities to eliminate a subset of these instances of undoing style, while preserving the semantics of the web application.

Prerequisites

- Firefox: https://www.mozilla.org/firefox/new/
- Firebug: https://addons.mozilla.org/nl/firefox/addon/firebug/
- npm: https://www.npmjs.com/
- The tool itself: https://leonardpunt.github.io/masterproject/

Setting it up

- Unzip the downloaded tool
- Install dependencies by running `npm install`
- Build the tool by running `grunt`
- Two build artefacts can be found in the folder `dist`:
  - `tool.min.js`: the tool with minified source code (for deployment)
  - `tool.js`: the tool with source code not minified (for debugging)

Running it

- Open Firefox
- Browse to the webpage you would like to analyse
- Open Firebug
- Paste contents of `tool.min.js` in Firebug's Command Line
- Click the **Run** button
- Now these two entry points are available in Firebug's Command Line:
  - `detect()` — Detects undoing style in CSS. Results of the analysis are saved to a text file. After the analysis is completed, the user is prompted to download the results (`detectionsX.txt`).
  - `detectAndRefactor()` — Detects undoing style in CSS and applies refactoring opportunities. Results of the analysis are saved to a text file. Refactoring opportunities are applied dynamically. After the analysis and refactoring is completed, the user is prompted to download the results. The results comprise:
    - Detected undoing style (`detectionsX.txt`)
    - Detected semantic changes (`errors.txt`)
    - Refactored stylesheets (name of original stylesheet)

Examples

The tool can be used to analyse a single state of a web application, as well as multiple states (or pages) of a web application. In this section we will demonstrate both cases. Furthermore, we will explain the output format.

Example 1: Single state

- Open the web application in the desired state. For instance, https://leonardpunt.github.io/masterproject/ has only one state.
- Run the tool as described in section Running it.

Example 2: Multiple states

- Capture the multiple states of the web application (e.g. by opening the web application in a desired state and saving it).
- Create a new HTML document, using this template:

```
<!DOCTYPE html>
<html>
<head>
  <title>[site name]</title>
</head>
<body>
  <iframe src="[some-state.html]"
          width="90%"
          height="300px"></iframe>
  <iframe src="[another-state.html]"
          width="90%"
          height="300px"></iframe>
</body>
</html>
```

- Open this new HTML document.
- Run tool as described in section Running it, only now pass the value `true` to both entry points. I.e. `detect(true)` and/or `detectAndRefactor(true)`.

Output

Example of output:

```
-- SMELL: 1 --
Smell: A-B(1)-A on margin-top
- initial: 0px (p)
- enclosed: 5px (#some-id)
- reset: 0px (#some-id.some-class)
- refactorable: true
- nodes: <p id="some-id" class="some-class">
```

The first line denotes the number of the undoing style pattern. The next line describes which pattern is found and on which CSS property, in this example we found the A^1B^1A pattern on the CSS property `margin-top`.

The next lines show the CSS selector which sets the style initially, the CSS selector(s) that are enclosed and the CSS selector that resets the style.

The last but one line shows if the pattern can be refactored. The last line shows the nodes that applied to the detected pattern.

Quality

All code conforms to the JSHint rules. Unit tests are included as well. Verify this by running grunt test.