Fixing the Web One Page at a Time, or Actually Implementing xkcd #37

Thomas Steiner
Universitat Politècnica
de Catalunya
Department LSI
08034 Barcelona, Spain
tsteiner@lsi.upc.edu

Ruben Verborgh and Rik Van de Walle
Ghent University – IBBT, ELIS
Multimedia Lab
9050 Ghent, Belgium
{ruben.verborgh,
rik.vandewalle}@ugent.be

ABSTRACT

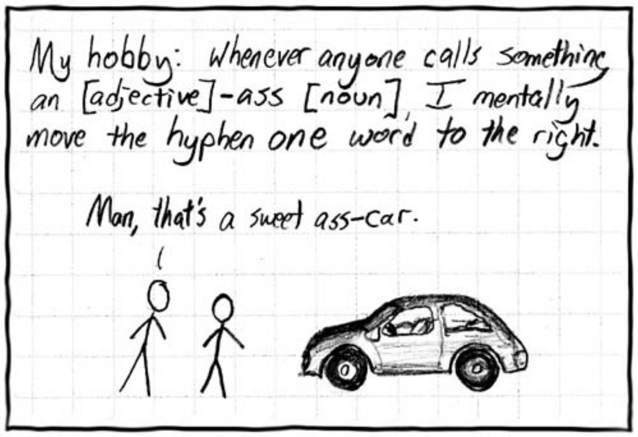

My hobby: whenever anyone calls something an [adjective]-ass [noun], I mentally move the hyphen one word to the right.
Man, that's a sweet ass-car.

Figure 1: xkcd #37 – I do this constantly [3].

1. INTRODUCTION

Albeit famous exceptions exist in form of Wikis, the Web today is still mostly a read-only experience. This leaves the Web content consumer exposed to all sorts of typographic cruelties, such as representing the ellipsis character '. . . ' with three single full stops "...", incorrect usage of a normal space where a non-breaking space would be preferred and even omission of the Oxford comma... While fighting the cause, namely sloppy Web authors, is like a fight against wind mills and certainly impossible to realize on Web scale, fighting the symptoms is a realistic option. Using client-side work-arounds, the Web can actually be fixed one page at a time. In this paper, we show how using browser extensions, part-of-speech tagging, and JavaScript DOM event listeners, the Web can be made a better place. On a related note, this is quite a sweet ass-abstract for a scientific paper, dude!

2. MOTIVATION

2.1 Typographic Annoyances

In this Subsection, we introduce common typographic annoyances with the objective of fixing them on the client-side.

Ellipsis character: Ellipsis (plural ellipses; from the Ancient Greek: ἔλλειψις, élleipsis, "omission" or "falling short") is a series of marks that usually indicate an intentional omission of a word, sentence, or whole section from the original text being quoted. An ellipsis can also be used to indicate an unfinished thought or, at the end of a sentence, a trailing off into silence (aposiopesis). The ellipsis character is commonly incorrectly represented by three full stops in a row due to the lack of a designated key on standard keyboards.

Oxford comma: The Oxford comma is the comma used immediately before a coordinating conjunction (usually "and" or "or", and sometimes "nor") preceding the final item in a list of three or more items, *e.g.*, "Portugal, Spain, and France". Opinions vary among writers and editors on the usage or avoidance of the serial comma. In American English, the serial comma is standard usage in non-journalistic writing that follows the Chicago Manual of Style. Journalists, however, usually follow the AP Stylebook, which advises against it, albeit the AP Stylebook errs here. There is no known valid excuse for not using the Oxford comma.

Non-breaking space: In computer-based text processing and digital typesetting, a non-breaking space is a variant of the space character that prevents an automatic line break (line wrap) at its position. Text-processing software typically assumes that an automatic line break may be inserted anywhere a space character occurs; a non-breaking space prevents this from happening (provided the software recognizes the character). For example, if the text "a house" will not quite fit at the end of a line, the software may insert a line break between "a" and "house". To avoid this undesirable behavior, the editor may choose to use a non-breaking space between "a" and "house". Due to its non-presence on standard keyboards, the non-breaking space typically gets represented by a normal white space character.

Appropriate dashes: Although they look similar to the untrained eye, different kinds of dashes come with substantial semantic differences. The em dash—being the longest of the family—indicates a break inside the normal sentence structure. Secondly, an en dash usually signifies a relationship between two compounds, such as a range (pages 1–3). Finally, a figure dash has a width similar to that of numerals and can be used for negative amounts (-1) or phone numbers (123–456–789). Unfortunately, the absence of dashes on keyboards and the only subtly different appearance makes users often unknowingly choose hyphens instead.

Typographic quotes and apostrophes: A final common typographic annoyance is using the wrong type of double quotes. Keyboards only have one symbol for both a beginning and an end quote, namely `"`. Commonly referred to as a typewriter or programmer's quote, it does not indicate to which part of the quotation it belongs. Therefore, "curved" quotes have been introduced as they are able to make this distinction. A similar situation is given for typewriter apostrophes ', which should be replaced by typographic apostrophes ' like in "don't", or "Randall Munroe's".

2.2 Further Use Cases

In this Subsection, we present further use cases where client-side fixing of Web pages can be considered useful.

xkcd #37: The Web comic xkcd in its episode #37 proposes the mental experiment of shifting the hyphen in word combinations of the form "[adjective]-ass [noun]" one word to the right, so that the resulting word combination reads "[adjective] ass-[noun]".

Emoticons: An emoticon is a pictorial representation of a facial expression using punctuation marks and letters, usually written to express a person's mood. Emoticons are often used to alert a responder to the tenor or temper of a statement, and can change and improve interpretation of plain text. Not emoticon-aware software unfortunately still has the tendency to break lines in the middle of an emoticon `:-(`. This can be avoided by the insertion of zero width no-break spaces between the characters that form the emoticon.

3. IMPLEMENTATION

In this Section, we detail the implementation and underlying technologies used. As motivated before, the only way to address typographic annoyances and advanced use cases is on the client-side. We therefore introduce browser extensions.

3.1 Browser Extensions

Browser extensions are small software programs that users can install to enrich their browsing experience with Web browsers. They are written using a combination of standard Web technologies, such as HTML, JavaScript, and CSS. There are several types of extensions; for this paper we focus on extensions based on so-called *content scripts*. Content scripts are JavaScript programs that run in the context of Web pages via dynamic code injection. By using the standard Document Object Model (DOM), they can modify details of Web pages. In the concrete case, we based our implementation on Google Chrome browser extensions[1], albeit given the lightweight architecture of browser extensions with the common building blocks JavaScript, HTML, and CSS, porting the extension to other browsers is possible.

3.2 Part-of-Speech Tagging

Simple typographic annoyances, like triple full stops instead of ellipses, can be easily fixed via regular expressions (here using JavaScript syntax):

```
"Lorem ipsum...".replace(/\.\.\./g, "...");
```

More complicated annoyances, like omission of the Oxford comma, not to mention xkcd #37, however, require basic

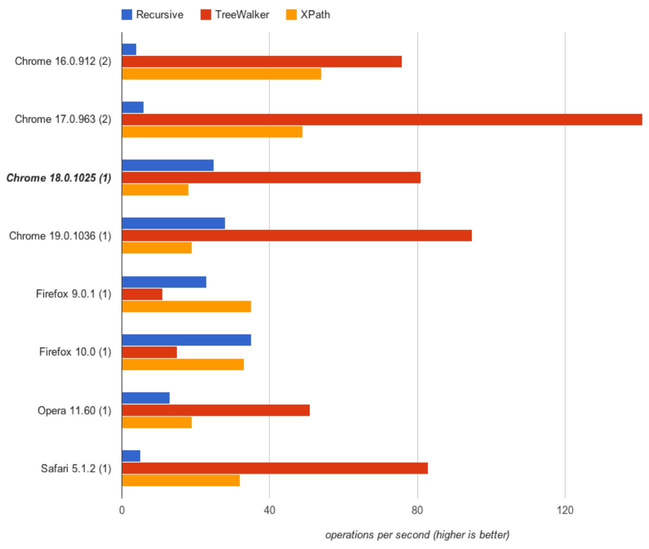


Figure 2: Comparing three different methods of obtaining all text nodes of a Web page. The winner is TreeWalker.

text understanding in order to only be fixed where adequate, *i.e.*, in lists of three or more items. On a similar note, it would harm the rules of xkcd #37 to replace "A bad-ass is afraid, but does it anyway" by "A bad ass-is afraid, but does it anyway", as "is" is not a noun.

A simple definition of part-of-speech tagging (POS) is the process of identifying words in a text as nouns, verbs, adjectives, adverbs, etc., based on both their definition, as well as their context. Our processing chain supports part-of-speech tagging for the English language via an open source JavaScript library called jspos[2], eventually based on Eric Brill's POS tagger [1]. The algorithm behind the Brill tagger assigns a tag to each word and later changes those tags using a set of predefined rules. If a word is known, the algorithm assigns the most frequent tag. Else, it first naively assigns the tag "noun" to the unknown word, and then applies the rules over and over, thereby changing incorrect tags, until a quite high accuracy is achieved.

In order for part-of-speech tagging to work, text fragments must be obtained from Web pages. In the following, we discuss techniques for obtaining text from Web pages.

3.3 Obtaining Text from Web Pages

We start this Subsection with a clarification of what we mean when we say text. We are not talking about the HTML source code of a Web page, but about the (in modern Web applications potentially dynamically generated) value of all nodes of the DOM tree with a JavaScript `nodeType` of integer 3, which corresponds to `Node.TEXT_NODE`. A naive approach would be to use `document.body.textContent`, however, this method neglects the context of the origin of the text. It would no longer be clear what text fragments belong together, *e.g.*, to the same paragraph. Instead, per-element text node analysis is necessary. There are three methods for achieving this: (i) selecting all text nodes via XPath,

[1]`http://code.google.com/chrome/extensions/`
[2]`http://code.google.com/p/jspos/`

(ii) recursively walking down the DOM tree, (iii) using the little-known DOM TreeWalker [2]. We have performed JavaScript performance tests in order to determine the fastest variant. The results of our tests can be seen in Figure 2. The DOM TreeWalker clearly outperforms all other methods. New results are dynamically added to the existing results by running the tests on a dedicated `jsPerf.com` page[3].

3.4 Listening on DOM Changes

The functionality so far only enables us to fix the static Web, *i.e.*, Web pages whose content remains unchanged during a browser session. However, an important share of modern Web pages makes use of dynamically retrieved information, for example with Ajax technologies. This means that a single text node manipulation cycle after the initial page has been loaded is insufficient.

Instead, we have to dynamically react on page changes by listening to DOM events [4]. Therefore, we add listeners for the `DOMCharacterDataModified` event, which occurs if the data of a text node is changed. Additionally, we monitor the `DOMSubtreeModified` event to watch when new elements are added to the DOM. If this is the case, we also inspect their contents for possible replacements.

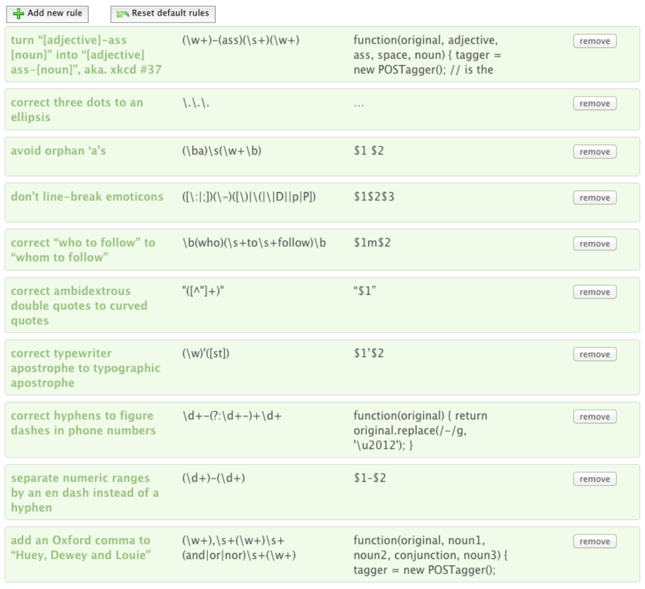
Finally, the `title` element deserves special attention. While it resides in the `head` element and thus out of the visible part of the DOM (in the `body` element), browsers usually display the title in a prominent place. For that reason, title changes are separately monitored by the `DOMSubtreeModified` event. The complete pseudocode for the implemented browser extension is given in Listing 1.

```
// Initial processing
for all text nodes of DOM tree as text node
  processRules(text node)
end for

// DOMNodeInserted Event Listener
on DOM node inserted (new node)
  for all text nodes of new node as text node
    processRules(text node)
  end for
end on DOM node inserted

// Helper function
function processRules(text node)
  for all rules as rule
    if rule is regular expression rule
      apply rule to text node
    else if rule is part-of-speech tagging rule
      apply part-of-speech tagging to text node
      apply rule to parsed text node
    end if
  end for
end function
```

Listing 1: Pseudocode for the browser extension's logic.

3.5 Practicability Considerations

The extension works best, if the user is not even aware of its presence—it should simply silently fix the Web. One important aspect therefore is smoothness and speed. The extension only gets active when everything on the page has rendered, *i.e.*, when the document is idle. On purpose we chose not to highlight any changes made by the extension, albeit implementing it would be trivial. A blacklist of HTML



turn "[adjective]-ass [noun]" into "[adjective] ass-[noun]", aka. xkcd #37	(\w+)-(ass)(\s+)(\w+)	function(original, adjective, ass, space, noun) { tagger = new POSTagger(); // is the	remove
correct three dots to an ellipsis	\.\.\.	…	remove
avoid orphan 'a's	(\ba)\s(\w+\b)	$1 $2	remove
don't line-break emoticons	([\:|;])(\-)([\)|\(|\\|D||p|P])	$1$2$3	remove
correct "who to follow" to "whom to follow"	\b(who)(\s+to\s+follow)\b	$1m$2	remove
correct ambidextrous double quotes to curved quotes	"([^"]+)"	"$1"	remove
correct typewriter apostrophe to typographic apostrophe	(\w)'([st])	$1'$2	remove
correct hyphens to figure dashes in phone numbers	\d+-(?:\d+-)+\d+	function(original) { return original.replace(/-/g, '\u2012'); }	remove
separate numeric ranges by an en dash instead of a hyphen	(\d+)-(\d+)	$1–$2	remove
add an Oxford comma to "Huey, Dewey and Louie"	(\w+),\s+(\w+)\s+(and|or|nor)\s+(\w+)	function(original, noun1, noun2, conjunction, noun3) { tagger = new POSTagger();	remove

Figure 3: Rules editor of the xkcd #37 extension.

tags (`input`, `code`, `textarea`, `pre`) ensures that no undesired corrections are being made. Accessibility of Web pages can be improved by a future version of the extension, *e.g.*, by annotating emoticons semantically. A further use case could be autocorrecting user input.

4. CONCLUSION

While the extension itself is ideally directly tested in a browser, Figure 3 shows the rules editor, which allows for new rules to be added, undesired rules to be deactivated, or existing rules to be modified. We have published our extension on the Chrome Web Store[4] and invite the reader to test it thoroughly. As a teaser, it is a very satisfying dumb ass-experience to read the Wikipedia page on the Oxford comma. Fun is to be had.

5. ACKNOWLEDGMENTS

We thank Christopher Blum (`@ChristopherBlum`) and Malte Ubl (`@cramforce`) for pointing us to DOM TreeWalker.

6. REFERENCES

[1] E. Brill. A simple rule-based part of speech tagger. In *Workshop on Speech and Natural Language*, pages 112–116, 1992.

[2] Mozilla Developer Network. Document Object Model (DOM) – document.createTreeWalker. `https://developer.mozilla.org/en/DOM/document.createTreeWalker`.

[3] R. Munroe. Hyphen. `http://xkcd.com/37/`.

[4] D. Schepers, J. Rossi, B. Höhrmann, P. Le Hégaret, and T. Pixley. Document Object Model (DOM) Level 3 Events Specification. W3C Working Draft, May 2011.

[3] `http://jsperf.com/obtain-all-text-nodes-of-a-web-page`

[4] `http://bit.ly/xkcd37`