Introduction

This is a review of a paper on a language-specific routines called Tutorons [1] that automatically generate context-relevant, on-demand micro- explanations of code to provide the missing scaffolding for unexplained code in tutorials, Q&As, and other online programming help. Tutorons automatically find pieces of code in web pages and augment these with micro-explanations: such explanations are context-relevant, describing only the syntactic elements that are present and important within a snippet, using domain-specific terms. They are also on-demand — they can describe confusing code found anywhere to provide just-in-time understanding. In the paper, they demonstrate techniques for generating natural language explanations through template instantiation, synthesizing code demonstrations by parse tree traversal, and building compound explanations of co-occurring options.

Processing stages

A. *Detection*

In the detection stage, a Tutoron should extract explainable regions from an HTML document using the language's lexicon and / or syntax. This can consist of four steps:

First, the Tutoron extracts blocks of code from HTML pages by selecting code-related elements. Each of our current Tutorons extracts blocks of code from some combination of <code>, <pre>, <div>, and <p> elements.

Second, it divides code blocks into candidate explainable regions based on language-dependent rules. CSS selectors and regular expressions can be detected as string literals in parsed Python or JavaScript code, requiring an initial parsing stage to detect these candidate regions. Commands like wget often occupy a line of a script, meaning that candidate regions can be found by splitting code blocks into lines.

Third, it passes candidate regions through a language-specific syntax checker to determine if it follows the grammar. Note that this can be combined with the parsing stage of the Tutoron processing pipeline.

Finally, a Tutoron may reduce false positives by filtering candidates to those containing tokens representative of the language. This is necessary when candidate regions are small and the lexicon and syntax of the language are lenient or broad. While a string like 'my_string' in a JavaScript program could represent a custom HTML element in a selector, it is more likely a string for some other purpose. Elements in a CSS selector more often than not have tag names defined in the HTML specification (e.g., p, div, or img).

It may happen that the same explainable region is detected by multiple Tutorons. For example, ‘div’ could be a CSS selector as well as a regular expression. While in practice the candidate regions of the current Tutorons differ enough that this is not a noticeable problem, they describe approaches to solve this problem in the discussion.

B. *Parsing*

Detected code snippets are parsed into some data structure in preparation for explanation. They have found two methods of building parsers to be useful. The first method is to introduce hooks into existing parsers to extract the results of parsing. This is useful when it is necessary to recognize a wide range of symbols and their semantics, such as handling the complex interplay of spaces, quotation marks, and redirects involved in parsing Unix commands.

The second method is to develop a custom parser for the language that supports a subset of the language. For CSS selectors, they wrote a 30-line ANTLR1 parser to gain control over the parse tree structure, allowing them to shape it to how they wanted to generate explanations. The resulting parser integrated nicely into existing code for the selector Tutoron.

C. *Explanation*

During the final stage, explanation, the Tutoron traverses the parse structure to generate explanations and demonstrations of the code. The implementation details of this stage are specific to the parse structure and the desired micro-explanation. They describe techniques to provide inspiration for future language-specific micro-explanations.

Conclusions

Programmers frequently turn to the web to solve coding problems and to find example code. Tutorons produce explanations for unexplained code that programmers find on the web. In their framework, Tutorons can produce effective micro-explanations when they leverage multiple representations, focus on dynamically generated content, build on existing documentation, and address common usage. They show that Tutorons for a collection of languages achieve around 80% precision detecting code examples, and improve programmers' ability to modify online code examples without referencing external documentation. Tutorons bring context-relevant, on-demand documentation to the code examples programmers encounter in online programming documentation.

References

[1] Andrew Head, Codanda Appachu, Marti A. Hearst, and Bjo¨rn Hartmann, "Tutorons: Generating Context-Relevant, On-Demand Explanations and Demonstrations of Online Code"