Automatic template extraction

Tom Rochette <tom.rochette@coreteks.org>

December 21, 2025 — 77e1b28a

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- To extract patterns, group them by starting character, then test how many have the same following character
- Grammar induction
- Compression
  - Compression can be a tool for automatic template extraction, however we would most likely want to priority semantics of the extracted template over better compression
- Diff/match/patch
- Fragment extraction, then wildcard pattern generation
- Lexer-like that will replace a whole sequence if it is already in the grammar instead of doing character by character replacement like sequitur

1 Overview

- Extract textual templates from any language (basically tries to find repetitions/patterns)
- Min/max length (characters)
- Discovery of syntax
- Hierarchical/meta extraction

2 Example

<...> is a placeholder (can be replaced/is variable)

2.1 If extraction

3 Prototype ideas/pseudo-code

- Create a dictionary of all seen characters
- Create a dictionary of characters -> index
- Define some sort of relative threshold for which to ignore patterns
- You have a single string, you want to extract patterns out of it
- You have two strings, you want to extract patterns out of them

4 Questions

- How to extract simple constructs such as if/elseif/else/while/do/for/foreach?

- How to compress aaaabbbb into an expanding aCb -> aaCbb -> aaaCbbb -> aaaabbbb vs AB -> aaaaB -> aaaabbbb
  - aaaabbbb -> aaaCbbb -> aaDbb -> aEb -> F
    * C := ab
    * D := aCb
    * E := aDb
    * F := aEb
    * C := aCb
      -> This is a context-free grammar
- Do we want to prioritize short rules such as S -> Sa such that they can be repeated many times, or rules that contains a lot of symbols such as S -> aSa
  - Probably want to minimize the number of rules/productions
  - Probably want to minimize the rule length
- From [^1]
  - p1: no pair of adjacent symbols appears more than once in the grammar;
  - p2: every rule is used more than once.
- How can we prefer `public function <>(<>) {<>}` over `} public function <>(<>) {`?
  - If we refer to an explicit grammar, we can give more weight to the first one because it is likely a construct/production in the grammar, while the second one is the concatenation of two productions

5 See also

6 References

- http://www.sequitur.info/
- Identifying Hierarchical Structure in Sequences: A linear-time algorithm
- https://en.wikipedia.org/wiki/Three-address_code
- https://en.wikipedia.org/wiki/Optimizing_compiler
- https://en.wikipedia.org/wiki/Intermediate_representation
- https://en.wikipedia.org/wiki/Abstract_syntax_tree