Clone Detection Using DMS® as a Universal Analysis Engine

Ira D. Baxter
Semantic Designs (SD), Inc.
idbaxter@semanticdesigns.com

Abstract

Practical clone detection using Abstract Syntax Trees [1] requires robust parsers for targeted source languages. A consistent theme of SD's DMS® Software Reengineering Toolkit [2] is amortization of construction cost of software engineering tools by use of shared infrastructure containing typical language processing services, such as parsing, prettyprinting, tree construction, control and data flow analysis, and source-to-source transformations. SD's CloneDR™ tool leverages DMS's facilities for use across multiple languages, for a variety of clone detection, reporting, and removal purposes. It also places demands on such infrastructure to meet its special needs, but such demands are typical of the needs of other program analysis and change tools. We intend to leverage the mutual dependency to grow both capabilities.

1. Status

The CloneDR [3] tool uses DMS language definition tools driving its GLR parsing engine as its source of abstract syntax trees, enabling its application to a wide variety of languages (C, C++, C#, Java, PHP, COBOL, Verilog, etc.) The generic clone detection engine is completely parameterized by the language definition. Consequently a CloneDR can be configured for each new DMS grammar in about an hour.

The CloneDR constructs reports about detected clone sets and their generic abstractions by using the DMS AST prettyprinter to normalize the format of the displayed clones to ease readability.

Batch clone removal for C and COBOL have been implemented using DMS transforms on the abstraction tree to generate an AST for a preprocessor macro and substituting a macro invocation at the clone site.

The parallel programming language, PARLANSE, underlying DMS is used by the generic detection algorithm to help minimize detection time. We plan to implement parsing many files in parallel soon.

2. Issues and Directions

Real parsing problems remain, such as dialects (handled reasonably by DMS), preprocessing (realistically requiring parsing of source code without expansion), and nested languages (requiring composition of parsing engines).

While $a+b$ is always syntactically a clone of $x+y$, one could construct more believable clones by insisting that clones process similar data types, e.g., both expressions are processing strings or numbers, but not a mixture. We expect to leverage DMS's symbol tables constructed by DMS language front ends to enable this.

For scale, faster clone detection is always desired. We are likely to reimplement CloneDR using suffix trees [4], but still leverage the parallelism.

Practical clone removal requires means to interactively select possible language-specific abstractions and carry out sophisticated lifting transformations. DMS's recent acquisition of sophisticated control and data flow analysis machinery [3] should enable this.

To do any clone removal, compiler-like parsing and analysis will surely be needed. This calls into question the real utility of string/token based clone detectors.

10. References

[1] Baxter, I., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L. "Clone Detection Using Abstract Syntax Trees". Proceedings of the International Conference on Software Maintenance, 1998, IEEE Press.

[2] Baxter, I., Pidgeon, P., and Mehlich, M. 2004. "DMS: Program Transformations for Practical Scalable Software Evolution". Proceedings of the International Conference on Software Engineering, 2004, IEEE Press.

[3] www.semanticdesigns.com. Company Website.

[4] Falke, R, Koschke, R. and Frenzel, P. "Empirical Evaluation of Clone Detection Using Syntax Suffix Trees". Empirical Software Engineering, 2008.