Clone Detection and Removal for Erlang/OTP within a Refactoring Environment

Huiqing Li
Computing Laboratory
University of Kent, UK
H.Li@kent.ac.uk

Simon Thompson
Computing Laboratory
University of Kent, UK
S.J.Thompson@kent.ac.uk

Abstract

This paper proposes a token and AST based hybrid approach to automatically detecting code clones in Erlang/OTP programs, underlying a collection of refactorings to support user-controlled automatic clone removal. Both the clone detector and the refactorings are integrated within Wrangler, the refactoring tool developed for Erlang/OTP.

1. Introduction

Duplicated code could be refactored out after having been introduced, but could also be avoided by first refactoring the existing code to make it more reusable, then reuse it without duplicating the code. Substantial research effort has been put into the detection and removal of clones from software systems; however, few such tools are available for functional programs, and there is a particular lack of tools that are integrated with existing programming environments.

Erlang/OTP is an industrial strength functional programming language with built-in support concurrency, communication, distribution, fault-tolerance, and dynamic code reloading [1]. This paper investigates the application of clone detection and removal techniques to Erlang/OTP programs within the refactoring context, proposes a token and annotated AST based hybrid approach to automatically detecting code clones across multiple modules, and describes three basic refactorings which together help to remove code clones under the user's control. Both the clone detector and the refactorings have been implemented within Wrangler [2]. Wrangler is a tool that supports interactive refactoring of Erlang/OTP programs. It is integrated with Emacs and now also with Eclipse. Wrangler itself is implemented in Erlang.

2. Wrangler's Clone Detection and Removal

Wrangler's clone detector is able to report code fragments in an Erlang project, either in a single module or across the whole project, that are syntactically identical after semantic preserving renaming of variables, also allowing for variations in literals, layout and comments. Syntactically, each of these code clones is a sequence of well-formed expressions or functions.

This approach makes use of both a token suffix tree and abstract syntax tree (AST) annotated with location and static semantic information. The use of the token suffix tree allows us to detect the initial clone candidates quickly, whereas use of the AST ensures that the tool only reports syntactically well-formed clone candidates. Furthermore, static semantic information annotated in the AST is used to check whether two code fragments can be refactored to each other by consistent renaming of variables and literals, and thus to ensure that the clones detected are actually removable.

Once a code clone set is found, it is possible to use refactorings in Wrangler to remove the clones. First, the duplicated code is *extracted* into a function, next it may be *generalised* to abstract over any literals. Finally, it is possible to step through all the instances of this function in the code base, deciding for each one whether or not to *fold* it into a function call. Three refactorings have been implemented to support the above process, and they are *function extraction*, *generalise a function definition*, and *fold expressions against a function definition*. These refactorings respect the importance of user intervention, and allow clones to be removed step by step under the programmer's control. Apart from removing code clones from legacy programs, these refactorings are also for programmers to use as part of their daily programming activities to avoid the introduction of code clones from the very beginning. While clone detectors shows where and how much code has been duplicated, our experiences show that tool support for clone removal is essential for clones to be removed, given the tedious and error-prone nature of manual refactoring.

In the future, we would like to investigate the use of clone detection and removal techniques to testing code. More details about Wrangler's clone detection and removal algorithm are reported in [2].

References

[1] J. Armstrong. *Programming Erlang*. Pragmatic Bookshelf, 2007.

[2] H. Li and S. Thompson. Clone Detection and Removal for Erlang/OTP within a Refactoring Environment. In *ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM'09)*, Savannah, Georgia, USA, January 2009.