OpenRefactory/C

An Infrastructure for Developing Program Transformations for C Programs

Munawar Hafiz
Auburn University
munawar@auburn.edu

Jeffrey Overbey
National Center for Supercomputing Applications
overbey2@illinois.edu

Abstract

This demonstration will provide an overview of OpenRefactory/C, an infrastructure for developing source-level program transformations for C programs. OpenRefactory/C is platform independent; however, the demonstration will be on the Eclipse platform. We will highlight the features of the infrastructure, outline the problems it solves, show the program analyses support that we have built for this infrastructure, and show traditional refactorings as well as advanced security-oriented program transformations that cannot be developed in any other C IDEs.

Categories and Subject Descriptors D.2.3 [*Software Engineering*]: Coding Tools and Techniques; D.2.7 [*Software Engineering*]: Distribution, Maintenance, and Enhancement

General Terms Languages, Security

Keywords Program Transformation, C, Preprocessor

1. The Problem

IDEs for modern languages, e.g., Java and C#, have support for maintenance and code evolution using automated refactorings. C, in spite of its popularity, has IDEs with a limited portfolio of program transformations, with limited static analysis capabilities, limited scalability, and limited applicability to real-world programs—particularly ones that make extensive use of the C preprocessor.

There are two major challenges that make it hard to build practical tools for transforming C programs.

(1) C programming IDEs ignore multiple configurations of C preprocessor because of its complexity; most IDEs provide program analysis and transformation support based on a single configuration. The resulting program transformations are inaccurate.

(2) IDEs for C do not support sophisticated static analyses even on preprocessed C code, e.g., Eclipse CDT only supports name binding, type analysis and limited control flow analysis. Without data flow analysis, it is impossible to implement any non-trivial program transformation. Preprocessors and multiple configurations make static analysis even more complicated.

2. The Demonstration

Existing tools have attempted to repurpose compiler- or IDE-based language infrastructures for refactoring. In contrast, OpenRefactory/C was built, from the beginning, with the goal of correctly refactoring C. It is based on a well-established infrastructural and API design, and it is being designed to support a sophisticated suite of static analyses while respecting the full semantics of the C preprocessor.

The demonstration will contain these parts.

(1) Problem explanation, and an illustration of the limitations of refactorings in current IDEs for C.
(2) Basic features of OpenRefactory/C highlighting the program analyses supported by the infrastructure.
(3) Advanced security-oriented program transformations [3, 4] that prevent various security vulnerabilities, most importantly integer overflow and buffer overflow vulnerabilities, that are implemented on OpenRefactory/C.
(4) The big picture and the plans ahead.

3. Features of OpenRefactory/C

The motivation behind OpenRefactory/C is our work on security-oriented program transformations [3]—complex transformations, that cannot be implemented by existing infrastructures with limited support for sophisticated analyses.

Our current infrastructure supports name binding analysis, type analysis, control flow analysis and static call-graph analysis. Name binding analysis information is encoded in the abstract syntax tree (AST). Any AST node representing a name reference can be queried to return the corresponding declaration node. Control flow and type information are computed as a query to the AST. Control flow analysis follows Morgenthaler's [7] algorithm of AST nodes dynamically computing which other AST nodes constitute its control flow successors and predecessors. Type analysis uses the results of name binding analysis to compute types of variables and functions. Call graph analysis uses the results of both the name binding and control flow analyses to annotate the AST with call graph edges. We have been working on integrating pointer analysis, specifically Anderson's points-to analysis [2], by interfacing with a points-to analysis implementation [6] based on Hardekopf's [5] algorithm.

4. Under the Hood

OpenRefactory/C is a plug-in for OpenRefactory, a framework for building source-level program analyses and transformations. OpenRefactory is so named because it is designed to be extensible both in terms of the refactorings it supports as well as the languages it can refactor. OpenRefactory/C adds C language support to OpenRefactory.

The refactoring infrastructure in OpenRefactory/C is based on the design of Photran [1], an Eclipse-based integrated development environment and refactoring tool for Fortran. Photran's refactoring infrastructure and source rewriting APIs were refined over the course of several years: Photran 8 contains 39 refactorings for Fortran programs, many of which were developed by third-party contributors. Thus, Photran's refactoring API has proven to be both general and reasonably easy for new contributors to learn, making it a good starting point for a C refactoring tool.

Like Photran, OpenRefactory/C's internal program representation is a rewritable abstract syntax tree generated by Ludwig [8]. The syntax tree is augmented with preprocessor information as described by Overbey, Michelotti, and Johnson [10]. Semantic checks are based on a differential precondition checking infrastructure [9]. Our infrastructure currently handles a *single preprocessor configuration*, i.e., it assumes that C Preprocessor macros take only one, fixed value; thus, it ignores some code in `#ifdef` regions.

Photran is based on Eclipse, but most of its underlying infrastructure is platform-independent. Our demonstration will also be based on Eclipse, but the infrastructure is platform-independent; it could be plugged in to other IDEs such as Visual Studio, or Vim.

5. Future Plans

The C preprocessor poses a particular challenge because most C programs use multiple preprocessor configurations: Through the use of macros and `#ifdef` directives, programmers vary what code is included in the executable, e.g., in debug vs. release configurations, $x86$ vs. $x86\_64$, etc.

Almost all existing C program transformation tools present in IDEs work on a single configuration of preprocessed code. While this is normal behavior for a compiler, and generally good enough for an IDE, it is insufficient for a refactoring tool. For example, a tool that applies a rename refactoring to a variable in one configuration but ignores other configurations may change the behavior of a program.

OpenRefactory/C will eventually support *multiple preprocessor configurations*. This means that it will be able to transform un-preprocessed source code, exactly as the programmer sees it, and will analyze and transform code with respect to *all possible* macro configurations.

References

[1] Photran - An Integrated Development Environment and Refactoring Tool for Fortran. http://www.eclipse.org/photran/.

[2] L. O. Andersen. *Program Analysis and Specialization for the C Programming Language*. PhD thesis, DIKU, University of Copenhagen, May 1994. (DIKU report 94/19).

[3] M. Hafiz. *Security On Demand*. PhD thesis, University of Illinois at Urbana-Champaign, 2010.

[4] M. Hafiz. An 'Explicit Type Enforcement' program transformation tool for preventing integer vulnerabiliites. In *Companion of OOPSLA '11*, pages 21–22. ACM, 2011.

[5] B. Hardekopf and C. Lin. The ant and the grasshopper: fast and accurate pointer analysis for millions of lines of code. In *Proceedings of the ACM Conference on Programming Language Design and Implementation, 2007*. ACM, 2007.

[6] M. Méndez-Lojo, A. Mathew, and K. Pingali. Parallel inclusion-based points-to analysis. In *OOPSLA*, 2010.

[7] J. Morgenthaler. *Static Analysis for a Software Transformation Tool*. PhD thesis, UCSD, 1997.

[8] J. Overbey and R. Johnson. Generating rewritable abstract syntax trees. In *Software Language Engineering: First International Conference, SLE 2008. Revised Selected Papers*, volume 5452 of *Lecture Notes in Computer Science*, pages 114–133, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] J. Overbey and R. Johnson. Differential precondition checking: A lightweight, reusable analysis for refactoring tools. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011*. IEEE, 2011.

[10] J. Overbey, M. D. Michelotti, and R. Johnson. Toward a language-agnostic, syntactic representation for preprocessed code. In *WRT '09: Proceedings of the 3rd Workshop on Refactoring Tools*.

Presenter Biography. Munawar Hafiz is an Assistant Professor at Auburn University, AL. He leads the program analysis and program transformation aspects of OpenRefactory/C, especially how complex program transformations can be built on this infrastructure [4]. Munawar has presented his research in various forms at previous SPLASHes, e.g., as part of a tutorial, as a poster, and as a finalist project in ACM student research competition.

Jeffrey Overbey, currently a postdoc at National Center for Supercomputing Applications (NCSA), is the chief designer of OpenRefactory/C infrastructure. He is co-lead of Photran [1], an open source project hosted by the Eclipse Foundation. It is widely used by the scientific community (approximately 20,000 users worldwide). Photran influences many design decisions in OpenRefactory/C.