Macquarie University
Department of Computing

PROJECT REPORT

Static Type Analysis of Dynamically Typed Programming Language

Author: Stepan Sindelar
Student ID: 43600220
Supervisor of the project: Matthew Roberts
Study programme: exchange student

Contents

1. Introduction

1.1 Problem Description

Three out of top ten programming languages in TIOBE index[1] are dynamically typed languages. One of the reasons for their popularity is that they are usually easier to use and suitable for quick prototyping. But at the same time the possibility to omit type information, which might be helpful during the early stages of a software project, can lead to more error prone code, and eventually to problems in later phases of the development and maintenance. Dynamic typing is also challenging for the compilers or interpreters designers. With the type information, a compiler is usually able to emit more efficient code.

Programmers are aware of the possible problems with the maintenance of dynamically typed code and they often document the type information in documentation comments. However, the correspondence of the documentation and the actual code is not checked, and moreover the compiler usually does not take any advantage of having type hints in the comments.

The PHP programming language is one of the mentioned popular dynamic languages and Phalanger is an implementation of a PHP compiler that compiles PHP code into the .NET intermediate code, which was developed at the Department of Software Engineering of the Charles University in Prague. A part of the Phalanger project is also an implementation of PHP tools for Visual Studio.

Because of its dynamic nature, PHP code is more difficult to analyse than code written in a statically typed language, especially if we want the analysis to be reasonably fast so that it can be used in everyday development. There is an ongoing research of the static analysis methods for many different families of programming languages, including dynamic languages. The problem this project is addressing is to adapt and apply those methods on a real world and widely used programming language PHP. The result is a library that is capable of performing a static analysis of PHP code and can be integrated into the Phalanger project. The library should allow to plug in any kind of analysis, for example constant propagation. However, the main goal is to provide a type analysis in order to discover possible type related errors and mismatches with the type information in the documentation comments and possibly to allow the compiler to emit more efficient code.

1.2 Implementation Constraints

More detailed description: usage of Phalanger front end, have to provide support for compiler back end and IDE (re-analysis of once analysed code), focus on object oriented PHP 5 style projects.

Leave this here or put it in the chapter about implementation?

1.3 Thesis structure

2. Static Analysis

More detailed description of what static analysis is (as opposed to for example explicit model checking, verification, etc.). Terminology: context sensitive, path sensitive, symbolic execution, abstract interpretation, etc.

Few possibilities here:

- *describe algorithms and terminology around static analysis, but this would be all a book work. Then custom modifications and adaptation of those would be described in chapter Implementation.*
- *or describe both here (static analysis in general + custom modifications).*
- *or just reference the literature and describe only custom modifications.*

2.1 Data Flow Analysis

2.1.1 Rapid Analysis

2.1.2 Abstract Domain

2.2 The PHP Programming Language

Or as a standalone chapter?

What makes static analysis in PHP harder than of statically typed language

what are the caveats of PHP we had to deal with:

arbitrary expressions in continue and break,

conditional declarations,

user defined auto-loading, ...

PHPDoc annotations and their significance for type analysis (especially for the bug hunting)

3. Existing Software

This project can be compared to other projects on more levels:

- RPython, Ecstatic (Ruby, [2]): comparison to approaches used for type analysis in other dynamic languages, e.g. Python, Ruby. Probably more interesting from the academic point of view.
- Weverca [3], Pixy[4]: comparison to other tools that analyse PHP, e.g. Weverca, Pixy, but those tools focus on different aims, namely security vulnerabilities through user input, XSS attacks, etc.
- NetBeans, Zend Studio: comparison to real world tools that analyze PHP for the same type of bugs (unused variable, use of unassigned variable, etc.), but those tools usually do not provide any documentation and any comparison would have to be based either on benchmarking and evaluation or on detailed study of their source code, which would be time consuming.
- HipHop: its type analysis is meant only for compiler back end, so it tries to find only one type for a variable, if that fails, it does not continue with analysis of that variable. (As far as I understood from the source codes, because there are no articles or documentation).
- Phantm [5]: very similar project, but uses Scala. What PHP versions it supports? Intra-procedural? Integration into a compiler/interpreter.

I would suggest deeper comparison only to Phantm and to mention other things only in a few summarizing paragraphs.

4. Implementation

specific constraints:

must use Phalanger front end and its AST data structures,

should be ready to be connected in between the Phalanger front end and back end as a middle end,

should be ready to be used continuously: interactively re-analyze updated code when used inside IDE,

overall description:

Control Flow graph construction

discussion of the choice of intermediate representation (or in fact, why we do not use any intermediate representation).

generic framework for Data Flow analyses

type analysis

5. Results

Analyses of well known OSS PHP projects (PhpUnit, Zend Framework, Wordpress, Symphony, Nettte) – few examples of discovered errors plus tables with data (as an appendix?). Possibly comparison to the results of other tools (Phantm).

6. Conslusion

6.1 Future Work

Branched Data Flow analysis, integer interval analysis, integration with the compiler back-end.

Bibliography

[1] "Tiobe index for march 2014." `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`. Accessed: 2014-09-03.

[2] M. Madsen, P. Sørensen, and K. Kristensen, *Ecstatic–type inference for Ruby using the cartesian product algorithm*. PhD thesis, Master Thesis, Jun, 2007.

[3] D. Hauzar and J. Kofron, "Hunting bugs inside web applications,"

[4] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities," in *Security and Privacy, 2006 IEEE Symposium on*, pp. 6–pp, IEEE, 2006.

[5] E. Kneuss, P. Suter, and V. Kuncak, "On using static analysis to detect type errors in php applications," tech. rep., 2010.

List of Tables

Attachments