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 The Problem

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 [2] is an implementation of a PHP compiler that compiles PHP code into the .NET intermediate code. Phalanger 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. Furthermore, in the future the library might be integrated into the Phalanger back-end.

1.2 Thesis structure

The following chapter describes the challenges connected with analysing source code written in the PHP programming language and also approaches to static analysis of source code in general. In chapter 3, we briefly discuss existing software of this kind. Chapter 4 provides more detailed description of our implementation. The analysis has been evaluated on several middle to large sized open source PHP projects and the results are presented in chapter 5.

2. Analysing PHP Code

The PHP programming language first appeared in 1995[3]. Over the years the language has involved and so have the ways how programmers were using it. This project focuses on PHP version 5.5[1] and the aim for the analysis is to work well on PHP source code written in an object oriented manner, using modern PHP patterns and idioms that are described later in this text. The analysis, however, should work reasonably good on any valid PHP code. We do not focus only on websites, but also on PHP libraries and frameworks that by themselves do not contain any PHP files that produce HTML or any other output for the user.

2.1 PHP semantics

This section describes some important parts of the semantics of the PHP programming language, especially those that represent a challenge for static analysis.

In PHP, local or global variables, object fields and function or method parameters are dynamically typed, which means that they can hold values of completely different types at different times of execution.

2.1.1 Local Variables

Local variables in PHP do not need to be declared explicitly. Instead the first usage of a variable is also its declaration. If a variable's value is used before the variable got any value assigned, then the interpreter generates a notice, however the execution continues and value `null` is used instead. A variable can get a value assigned to it when it appears on a left hand side of an assignment or when a reference to that variable is created, in which case it gets value `null`. Note: references are discussed in one of the following subsections.

The scope of a local variable is always its parent function not the parent code block as in other languages like C or Java. So in the following example, the usage of variable `$y` at the end of the function can generate uninitialized variable notice, however, if `$x` was equal to `3`, `$y` will have a value although it was declared in the nested code block.

2.1.2 Global and Local Scope

2.1.3 Closures

PHP also supports anonymous functions. An anonymous function has its own scope as any other function and its local variables are not visible to the scope where it was declared

2.1.4 Interesting Control Flow Structures

Arbitrary expressions in continue and break.

[1] From this point, if the PHP version is not stated explicitly, it is implicitly 5.5.

2.1.5 Conditional Declarations

2.1.6 Auto-loading

2.1.7 PHPDoc Annotations

2.2 Static Code 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.

2.2.1 Data Flow Analysis

2.2.2 Rapid Analysis

2.2.3 Abstract Domain

3. Existing Software

3.1 Phantm[4]

3.2 HipHop Type Inference for Hack

3.3 Weverca: Web Verification Tool[5]

4. Implementation

4.1 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,

4.2 Overall Design

4.3 Control Flow Graph and Data-flow Analysis

- 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
- support for re-analysing pieces of code

4.4 Tables

Dependencies handling.

4.5 Type Analysis

- Type Information representation
- Data Flow representation
- AST annotations

5. Results

The project has been evaluated on the following open source PHP frameworks and websites.

- PhpUnit: a port of JUnit unit testing framework for PHP.
- Zend Framework: popular general purpose PHP framework.
- Nette: another popular PHP framework for building websites.
- WordPress: on of the most popular content management systems.
- Drupal: another popular content management system.

An evaluation always started with downloading a git repository with the latest source code of given PHP project. Than the analysis was run and all the discovered errors were rectified and recorded as commits in the git repository.

5.1 PhpUnit

5.2 Zend Framework

5.3 Nette

5.4 WordPress

5.5 Drupal

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] J. Benda, T. Matousek, and L. Prosek, "Phalanger: Compiling and running php applications on the microsoft .net platform," *.NET Technologies 2006*, 2006.

[3] "Php: History of php." http://www.php.net/manual/en/history.php.php. Accessed: 2014-02-04.

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

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

List of Tables

Attachments